Introduction tutorial to ShExML

Introduction

Shape Expressions Mapping Language (ShExML) is a Domain Specific Language based on Shape Expressions (ShEx) that allows the translation and homogenisation of heterogeneous data. Through this language the users can merge data sources in different formats in a single RDF Turtle output, creating a simple script. This avoids the creation of *ad-hoc* solutions for every data source and the consequent data homogeniser.

Structure

A ShExML script is constituted by two fundamental parts: declarations and *shapes*. A declaration is an expression that is linked to a variable name, as in the programming languages where a variable can be defined by a literal value or a function. The shapes, by its side, define the output shape (in ShEx they define the expected shape of the entity to be validated) allowing to use previously defined variables to obtain the subjects and objects values for the RDF triples. Declarations are divided into: *prefix*, *source*, *iterator* y *expression*. The *prefix* allows to declare prefixes with the aim to define the different IRIs that will be used along the script and in the RDF output. Prefixes are used as in Turtle and SPARQL. A *source* allows to define a data source by means of a URL. The *iterator* allows to define at which point the algorithm must iterate because many entities are present. Finally, the *expression* allows to use former declarations to compose and transform the data.

Step by step example

In this example we are going to see how a script can be constructed to merge two data sources (JSON and XML) in a single RDF output. The example will be constructed step by step so that in the end we can have the whole view of the created script.

Data sources

The data sources that we are going to use are two: a set of films in JSON and another set of films in XML. The content of these files can be consulted following the links in Table 1. Each file defines two different films with their corresponding attributes: id, name, year, country and directors.

Table 1: Links to the data sources

Format	Link
JSON	https://rawgit.com/herminiogg/ShExML/master/src/test/resources/films.json
XML	https://rawgit.com/herminiogg/ShExML/master/src/test/resources/films.xml

Defining prefixes

To define a prefix that will be used in the script and in the RDF output we use the statement showed in Snippet 1. We use the keyword PREFIX followed by the variable name that we want to use e.g., ex:, schema:, rdfs:, etc. The value for this variable is enclosed between the symbols < and >.

```
PREFIX ex: <http://example.com>
```

Snippet 1: Prefix declaration

Defining data sources

To define a data source we use the keyword SOURCE as it can be seen in Snippet 2. After the keyword the variable name is declared and, enclosed between the symbols < and >, the URL to the data source.

```
SOURCE films_xml_file
<https://rawgit.com/herminiogg/ShExML/master/src/test/resources/films.xml>
```

Snippet 2: Data source declaration

Defining iterators

Iterators allow us to define when, inside a file, it is necessary to iterate because more than one entity is present. This is the case of the films inside the two previously mentioned files. Thus, it will be necessary to define in which part the algorithm must iterator and which elements we want to take. This is achieved in ShExML by the keyword ITERATOR that allows to include a query in XPath or JSONPath. An example of this instruction can be seen in Snippet 3, where the XPath query is enclosed between symbols < and >. In addition to the iterator definition, the different fields that will be used later are defined. These fields are defined using XPath but without the navigational element '/' that is added automatically by the ShExML engine.

```
ITERATOR film_xml <xpath: //film> {
   FIELD id <@id>
   FIELD name <name>
   FIELD year <year>
   FIELD country <country>
   FIELD directors <directors/director>
}
```

Snippet 3: Iterator and fields declaration

Defining expressions

Expressions allow us to use former declarations to be able to process an iterator over a data source, merge the results of various iterators or to transform these results. Navigation through these elements is made using the '.' operator. In this manner, we can access to an iterator field using the syntax shown in Snippet 4. Apart from that, it is also possible to apply an iterator without accessing the fields. This will cause that the result will be a set of results (see full example).

```
EXPRESSION films_ids <films_xml_file.film_xml.id>
```

Snippet 4: Expression declaration

Shapes

As we have introduced earlier in this document, a *shape* is the mechanism to define the data format that we want to generate. Thus, we will have a *shape* name that will be used to link different shapes, an expression to generate subjects and a set of predicates and objects. Inside this set of predicates and objects, predicates will be constituted by a prefix and a termination (like in Turtle) and the objects will be a prefix and a termination, a prefix and an expression or just an expression. The expression will in charge of generating the different values for the objects. The full syntax can be seen in Snippet 5 where it can be seen how in :name a literal value is generated and in :year a URI is generated when the prefix in place before the expression evaluation.

```
:Films :[films_ids] {
```

```
   :name [films_names] ;
   :year :[films_years] ;
   :country [films_countries] ;
   :director [films_directors] ;
}
```

Snippet 5: Films Shape example

Full example

In the following example we can see how to merge the two previously mentioned files and how an iterator can be used to fill a *shape*. It is worth to highlight that in this example we can see how instead of using unions of fields, two iterators are merged that are then extracted in the *shape*. This makes the language more modular.

```
PREFIX : <http://example.com/>
SOURCE films_xml_file
<https://rawgit.com/herminiogg/ShExML/master/src/test/resources/films.xml>
SOURCE films_json_file
<https://rawgit.com/herminiogg/ShExML/master/src/test/resources/films.json>
ITERATOR film_xml <xpath: //film> {
   FIELD id <@id>
   FIELD name <name>
   FIELD year <year>
   FIELD country <country>
   FIELD directors <directors/director>
}
ITERATOR film_json <jsonpath: $.films[*]> {
   FIELD id <id>
   FIELD name <name>
   FIELD year <year>
   FIELD country <country>
   FIELD directors <director>
}
EXPRESSION films <films_xml_file.film_xml UNION films_json_file.film_json>

:Films :[films.id] {
   :name [films.name] ;
   :year [films.year] ;
   :country [films.country] ;
   :director [films.directors] ;
}
```

Snippet 6: ShExML script for films

```
@prefix :     <http://example.com/> .

:4     :country   "USA" ;
       :director  "Jonathan Nolan" , "Christopher Nolan" ;
       :name      "The Prestige" ;
       :year      2006 .

:3     :country   "USA" ;
       :director  "Christopher Nolan" ;
       :name      "Inception" ;
```

```
      :year     2010 .

:2     :country   "USA" ;
      :director  "Jonathan Nolan" , "Christopher Nolan" ;
      :name      "Interstellar" ;
      :year      2014 .

:1     :country   "USA" ;
      :director  "Christopher Nolan" ;
      :name      "Dunkirk" ;
      :year      2017 .
```

Snippet 7: Turtle result after applying the ShExML script